Problem 6: Find a Plan to Leave the Wumpus World

AI1SysProj 2021/2022

Topic:	Planning
Due on:	March 29, 2022
Version from:	February 17, 2022

1 Task Summary

Your agent is trapped in the Wumpus world. Find a way for your agent to leave it using a PDDL planner. The assignment repository [A6] contains maps for you to solve.

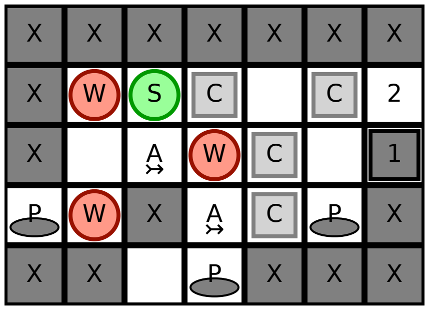

Figure 1: Example map of the Wumpus world.

2 The Wumpus World

Maps of the Wumpus world are encoded as a text file, where each line corresponds to a row of cells and each character describes the properties of an individual cell. Here is an example map:

```
XXXXXXX
XWSC␣C2
X␣AWC␣1
PWXACPX
XX␣PXXX
```

Cells can be blocked by e.g. walls (X) or a wumpus (W). Section 2.2 describes the different cells in more detail. Figure 1 shows a visualization of the example map.

2.1 Format of a Plan

Your plan is a sequence of actions. After performing those actions, your agent should be "off the map". Each action is of the form `[ACTION] [DIRECTION]`, where `[DIRECTION]` is one of `left`, `right`, `up` or `down`. The possible actions are:

- `walk`: The agent walks to the neighboring cell in the indicated direction. If there is no cell in that direction, the agent has left the map (which is the goal). Whether you are allowed to walk into a cell depends of course on the map – for example, you cannot walk into a wall. Section 2.2 has more details.
- `shoot`: Shoot a Wumpus. This requires a Wumpus to be in the neighboring cell in the indicated direction. You also must have at least one arrow. Afterwards, you have one arrow less and the Wumpus is gone.

Note that these actions do not have to correspond to the actions that you use for the planning itself.

2.2 Cell Properties

As mentioned earlier, the properties of each cell are summarized by a single character:

- `S`: The starting position of the agent. Otherwise the cell is empty.
- `␣` (a single whitespace): An empty cell.
- `X`: A wall. Your agent cannot walk into a cell with a wall.
- `W`: A wumpus. Your agent cannot walk into a cell with a Wumpus. However, the Wumpus can be shot with an arrow. Afterwards, the cell is free.
- `A`: An arrow. If you walk into a cell with an arrow, you automatically pick it up (and have one more arrow). Afterwards, the cell is empty.
- `P`: A pit. Your agent cannot walk into a cell with a pit. However, a pit can be filled by a crate (see next item).
- `C`: A crate. Your agent can only walk into a cell with a crate if the crate can be pushed into the next cell in the specified direction. If the next cell has a wall, a Wumpus, another crate or a door, it cannot be pushed. If the next cell is empty, the crate will be moved to the empty cell. If the next cell has a pit, the pit is filled with the crate and you can from now on treat it like an empty cell. If the next cell has an arrow, the crate can still be pushed into the cell – and if you then push the crate from that cell, you will retrieve the arrow.
- `1`, `2`: A (sliding) door. It is always made of two neighboring cells: one marked as `1`

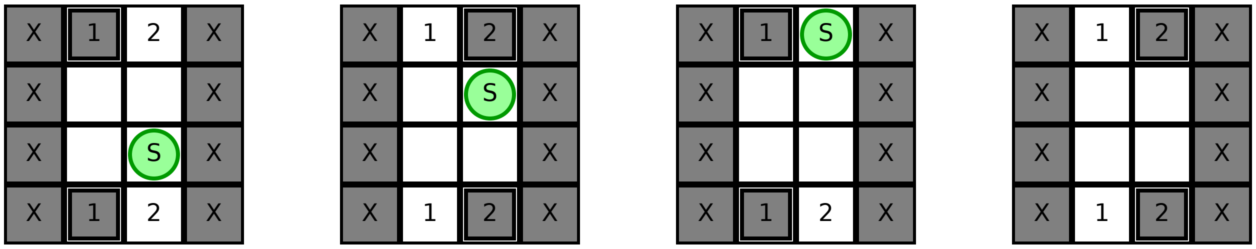

Figure 2: In this case, the map can be left by repeating the `walk up` action three times (we keep marking the agent location with an `S`). The door at the bottom cannot be used.

and one marked as `2`. Initially, the sliding door is blocking the cell marked as `1`. For every action, it switches the cell it is blocking. Figure 2 shows an example. If you are in a door, you must leave it the next move.

3 Planners

You should use a PDDL planner for this problem. PDDL planners require two files: a domain file, which we will call `wumpus.pddl`, and a problem file, which we will call `map.pddl`. The domain file should describe the rules of the Wumpus world in general and the problem file specifies the details of a particular map.

You can use the `fast-downward` planner or `pyperplan` (see below). If you want to use a different planner, please ask in the problem channel first.

3.1 `fast-downward` (recommended)

`fast-downward` [FD] is a heavily optimized planner that was quite successful in the the international planning competition (IPC). You can build it yourself (which worked very well for me) or run it using docker.

3.1.1 Building and Running `fast-downward` yourself

After cloning the repository, you can build `fast-downward` with

```
python3 build_configs.py
python3 build.py
```

Then you should be able to run it with

```
python3 fast-downward.py --alias lama-first \
    --plan-file /path/for/outputfile \
    /path/to/wumpus.pddl \
    /path/to/map.pddl
```

You can pick other modes than `lama-first`. Note that the `lama-first` mode does not attempt to find an optimal plan (i.e. a plan with a minimal number of actions).

3.1.2 Using Docker

If you have docker installed, you can simply run `fast-downward` with

```
docker run --rm -v "/path/to/my/files:/files" \
    aibasel/downward --alias lama-first \
    --plan-file /files/outputfile \
    /files/wumpus.pddl /files/map.pddl
```

The first line maps the directory `/path/to/my/files` to the docker volume `/files`. That means that `/path/to/my/files/xyz` on your machine will correspond to `/files/xyz`.

3.2 `pyperplan`

`pyperplan` [Alk+20] is implemented in Python and used as a teaching tool. As such, it is not heavily optimized but it should be perfectly sufficient for this assignment. The README file on github provides good instructions for installation and use.

3.3 Web Planner

Web Planner [Mag+17; WP] offers a web interface for editing PDDL files and visualizing the search space. As a website, it is not an alternative to `fast-downward` or `pyperplan`, but it might help you getting started with PDDL.

4 What to Submit

You should submit

- All your code and a README explaining how to use it for finding a plan for map. Your README should also state what planner you used (there are some differences in the PDDL files they support).

- Your domain file (`wumpus.pddl`).
- Solutions to all the example maps in the assignment repository [A6]. Concretely, you should submit for every map `mapXY.txt` the problem file `mapXY.pddl`, the solution of the PDDL problem `mapXY.pddl.soln`, and the sequence of `walk` and `shoot` actions to leave the map (Section 2.1) listed line by line in a file `mapXY.solution`.

5 Random Tips

- Don't try to support all cell types at once. Instead, it might be easier to initially only support `S`, `␣` and `X`. You can make test worlds to see if your approach works. The next types to support would probably be `C`, followed by `P` and then `A` and `W`. The doors (`1` and `2`) are presumably the hardest to support.
- The actions in your PDDL file don't have to directly correspond to the "solution" actions described described in Section 2.1 (e.g. you might want a separate PDDL action for pushing a crate). The only condition is that the "solution" actions can be easily generated from the PDDL actions.
- You can verify solutions (including for worlds you made yourself) with the verification script provided in the assignment repository [A6].
- Have fun!

References

[A6] *Assignment 6*. URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2122/planning/assignment` (visited on 02/15/2022).

[Alk+20] Yusra Alkhazraji et al. *Pyperplan*. `https://doi.org/10.5281/zenodo.3700819`. 2020. DOI: `10.5281/zenodo.3700819`. URL: `https://doi.org/10.5281/zenodo.3700819`.

[FD] *Fast Downward*. URL: `https://www.fast-downward.org/` (visited on 02/15/2022).

[Mag+17] Maurício C Magnaguagno et al. "WEB PLANNER: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search". In: *Proceedings of the Workshop on User Interfaces and Scheduling and Planning, UISP*. 2017, pp. 32–38.

[WP] *Web Planner*. URL: `http://web-planner.herokuapp.com/` (visited on 02/17/2022).

A Solution to the Example Problem

A solution to the example problem from Section 2 would be:

```
walk down
shoot right
walk right
walk down
walk right
walk right
walk up
walk left
walk up
walk left
walk down
walk down
walk down
walk down
```

which could be visualized as